Applications Rely on Geo-Replicated Storage

• **Fault tolerant**: data is safe despite failures
Applications Rely on Geo-Replicated Storage

- **Fault tolerant**: data is safe despite failures
- **Linearizable**: intuitive for application developers
Linearizable Replicated Storage Systems

etcd

Cockroach DB

Cloud Spanner

Azure

The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, Google Inc.
Status Quo: Consensus or Shared Registers

- Given the desire for **fault tolerance** and **linearizability**

<table>
<thead>
<tr>
<th></th>
<th>Consensus</th>
<th>Shared Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Synchronization</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Low Read Tail Latency</td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Unify consensus and shared registers?**
Consensus & State Machine Replication (SMR)

• Generic interface: $\text{Command}(c(.)$)
• Stable ordering: all preceding log positions are assigned commands

\[ C_1 \quad C_2 \quad C_3 \quad C_4 \]
Consensus & State Machine Replication (SMR)

• Generic interface: \texttt{Command(c(.))}
• \textbf{Stable ordering}: all preceding log positions are assigned commands
• Used in etcd, CockroachDB, Spanner, Azure Storage, Chubby
SMR Requires Stable Order

- Allow for strong synchronization primitives like read-modify-writes
- High tail latency in practice (e.g., by serializing through a leader)

<table>
<thead>
<tr>
<th>Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Synchronization</td>
</tr>
<tr>
<td>Low Read Tail Latency</td>
</tr>
</tbody>
</table>
Shared Registers

• Simple interface: `Read() / Write(v)`
• **Unstable ordering**: total order without pre-defined positions
Shared Registers

• Simple interface: Read()/Write(v)
• Unstable ordering: total order without pre-defined positions
Shared Registers

• Simple interface: `Read() / Write(v)`
• Unstable ordering: total order without pre-defined positions
• Similar to Cassandra, Dynamo, Riak

\[ W_1 < W_2 < W_3 < W_5 < W_4 \]
Shared Registers Use Unstable Order

- Cannot implement strong synchronization primitives [Herlihy91]
- Flexibility of unstable order provides favorable tail latency

<table>
<thead>
<tr>
<th></th>
<th>Consensus</th>
<th>Shared Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Synchronization</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Low Read Tail Latency</td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>
Shared Objects: Interface for Unification

• Interface: \texttt{Read(v)/Write(v)/RMW(f(.))}
• \texttt{RMW(f(.))} \rightarrow \text{read base } v, \text{ compute new value } f(v), \text{ write } f(v)
• Examples: etcd, Redis, BigTable

RMWs with low read tail latency?
Consensus-after-Register Timestamps (Carstamps)

Unstable Order

Stable Order

W₁ < W₂ < W₃ < W₄

rmw₁ < rmw₃ < rmw₄

rmw₂
Consensus-after-Register Timestamps (Carstamps)

Unstable Order

W_1 < W_2 < W_3 < W_5 < W_4 < W_6

Stable Order

rmw_1 < rmw_3 < rmw_4

rmw_2
Carstamps

• Tuple with three fields: \((ts, id, rmwc)\)
• \(ts\) and \(id\) basis for **unstable ordering of writes**
• \(rmwc\) is set to 1 greater than \(rmwc\) of base to ensure **stable ordering**

\[
\begin{align*}
(3,1,0) & < W_1 \text{ \hspace{1cm} } W_2 (4,1,0) \\
(3,1,1) & \text{ \hspace{1cm} } rmw_1 \text{ \hspace{1cm} } rmw_3 (4,1,1) \\
(3,1,2) & \text{ \hspace{1cm} } rmw_2
\end{align*}
\]
Gryff Unifies Consensus and Shared Registers

• Only uses consensus when necessary, for strong synchronization

<table>
<thead>
<tr>
<th></th>
<th>Consensus</th>
<th>Shared Registers</th>
<th>Gryff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Synchronization</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Low Read Tail Latency</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Gryff Design


• Modifications needed for safety:
  • Carstamps for proper ordering
  • Synchronous Commit phase for rmws

• Modifications for better read tail latency:
  • Early termination for reads (fast path)
  • Proxy optimization for reads (fast path more often)

See the paper for details!
Gryff in Action
Gryff in Action

(2,3,0)  (1,0,0)  (2,3,0)
Gryff in Action

$w_1 \rightarrow (3,1,0)$

$\text{Write1Reply} (1,0,0)$

$\text{Write1Reply} (2,3,0)$

$\text{Writes always terminate in 2 phases}$
Gryff in Action

Executed (3,1,1)

rmw₁ → (3,1,1)

C₁

Writes always terminate in 2 phases
RMW carstamps directly after base

(3,1,1) (3,1,1) (2,3,0)
Gryff in Action

$c_1 \rightarrow (3,1,1)$

Read1Reply $(3,1,1)$

$c_2$

- Writes always terminate in 2 phases
- RMW carstamps directly after base
- Reads often terminate in 1 phase
Evaluation

Relative to state-of-the-art-consensus protocols:
1. How do Gryff’s read/write protocols affect read tail latency?
2. What is the latency distribution of Gryff’s reads, writes, and rmws?
3. What maximum throughput does Gryff achieve?
4. How does Gryff perform in tail-at-scale workloads?
Evaluation

Relative to state-of-the-art-consensus protocols:

1. How do Gryff’s read/write protocols affect **read tail latency**?

2. What is the latency distribution of Gryff’s reads, writes, and rmws?

3. What **maximum throughput** does Gryff achieve?

4. How does Gryff perform in **tail-at-scale** workloads?
Evaluation Setup

• Geo-replication with 3 regions

• Baselines: MultiPaxos (industry standard), EPaxos (leaderless)
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)

MultiPaxos
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)

- MultiPaxos

serializing through far-away leader
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)

Fraction of Reads

Latency (ms)

- Delaying reads that conflict with concurrent writes

MultiPaxos
EPaxos
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)

- MultiPaxos
- EPaxos
- Gryff
Read Tail Latency (94.5% R, 4.5% W, 1% RMW, 25% Conflicts)

1 round to nearest majority in tail
Summary

• Consensus: strong synchronization w/ high tail latency
  Shared registers: low tail latency w/o strong synchronization

• **Carstamps** stably order read-modify-writes within a more efficient unstable order for reads and writes

• **Gryff** unifies an optimized shared register protocol with a state-of-the-art consensus protocol using carstamps

• Gryff provides strong synchronization w/ low read tail latency
Image Attribution

• Griffin by Delapouite / CC BY 3.0 Unported (modified)
• etcd
• CockroachDB
• Spanner by Google / CC BY 4.0